A Counting model for software reliability analysis

James Ledoux, Gerardo Rubino

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

A Counting Model for Software Reliability Analysis

James Ledoux, Gerardo Rubino

N° 2060

Juillet 1993

PROGRAMME 1

Architectures parallèles,
bases de données,
réseaux et systèmes distribués

ISSN 0249-6399

INRIA
RENNES

A Counting Model for Software Reliability Analysis

James Ledoux*, Gerardo Rubino*

Programme 1 — Architectures parallèles, bases de données, réseaux
et systèmes distribués
Projet Model

Rapport de recherche n ° 2060 — Juillet 1993 — 17 pages

Abstract: Structural reliability models suffer from some drawbacks mainly related to the assumptions necessary to apply them. We propose a model which intends to overcome some of these limitations appearing in previous published works. We discuss on an approach that can make more realistic the usual Markovian assumptions when considering the structural modeling of the execution process. Moreover, we define a failure process allowing to model different and general situations and we give analytic results and algorithmic methods to perform the corresponding quantitative evaluations. We show that our model and the obtained results include as particular cases some previous proposed models. We also discuss about some interesting asymptotic properties of the model.

Key-words: Software Reliability, Markov Processes, Counting Processes, Uniformization Technique.

(Résumé : tsvp)

This work was partially supported by the grant 290C2010031305061 of the French Region Bretagne. Part of this report will be presented in the 2nd IASTED International Conference on Reliability, Quality Control and Risk Assessment which is to be held at Cambridge, Massachusetts, on Octobre 13–15 1993.

*{ledoux}{rubino}@irisa.fr

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : (33) 99 84 71 00 – Télécopie : (33) 99 38 38 32

Le processus de comptage d'un modèle de fiabilité du logiciel

Résumé : Les hypothèses de modélisation retenues dans l'évaluation structurelle de la fiabilité du logiciel sont généralement considérées comme trop réductrices. Le modèle présenté dans ce rapport tente de répondre à certaines de ces limitations. En particulier, nous proposons une approche simple pour rendre plus réalistes les usuelles hypothèses markoviennes du processus d'exécution du logiciel. Nous construisons ensuite un processus général de défaillance et nous donnons les outils analytiques et algorithmiques nécessaires à son évaluation. Ce modèle inclut comme cas particuliers certains modèles connus. Nous discutons également quelques unes de ses propriétés asymptotiques.

Mots-clé : Fiabilité du logiciel, processus de Markov, processus de comptage, technique d'uniformisation.

Contents

Main notation

- $\mathcal{M} = \{1, 2, \ldots, M\}$: the set of components;
- X_t: the active component at time t, for a failure-free system ($X = (X_t)_{t \geq 0}$ is the *execution process)*;
- X_t^*: the active component at time t, when failures are taken into account;
- N_t: the number of failures up to time t;
- $\alpha = (\alpha_1, \ldots, \alpha_M)$ where $\alpha_i = \mathbb{P}\{$ execution starts at component $i\}$;
- $Q = (q(i,j))_{i,j \in \mathcal{M}}$ where $q(i,j)$ is the transition rate of the execution process (X_t), from component i to component j;
- λ_i: primary failure rate when the active component is i;
- $\lambda(i,j)$: occurrence probability of a primary failure during a control transfer from component i to component j;
- $\alpha(i,j)$: probability of restart in component j given that the component i has provoked a break;
- μ_i (resp. $\mu(i,j)$) has the same meaning as λ_i (resp. $\lambda(i,j)$) for the secondary failure process.

1 Introduction

Software reliability (and more generally, software dependability) has become a central engineering concept. The increasing complexity of software has led to the development of methodologies that allow to handle, in a probabilistic framework, some quality aspects of the products, as it has been done for a long time for hardware. The fact that software reliability is a more recent area than hardware reliability is not the only difference between them: there are some deep particularities with software systems having strong influences on the concepts and techniques involved in their reliability analysis (for details, see [4],[8]).

Basically, there are two types of software models. The first class is composed by the so-called "black-box" models, in which the system is seen as a whole and only its interactions with the external world are considered. Typical data associated with this approach is the sequence of failure instants or simply of number of failures observed during a fixed period. Most of the efforts of the research community has been (and is) done in this context. In general, some parametric model of the system is chosen and the observations of the failure times (or of the number of failures) are used to adjust the parameter values. The main common feature of most of the numerous black-box models that have been proposed is their ability to capture the stochastic phenomenon of *reliability growth* [4],[24]. As a consequence, they are particularly well adapted to the first phases of the development process, where this phenomenon is in general significant.

Informally, the complexity of a software system is roughly proportional to its size. Often, large systems exhibit interactions between components having very different workload and failure behaviors. This is particularly important in the operational phase where failures occur in principle less frequently and where the action of the environment is quite different as, for instance, during the tests phase. This situation is better handled by the second approach to software reliability, known as *structural* or "white-box". Here, the model includes information about its internal structure. The counterpart of this is, of course, the fact that more sophisticated data is needed to use it. With respect to the black box approach, just a few papers have been published on structural software reliability models. A representative sample is [12],[13],[2],[9],[10],[15],[7],[1]. This paper adopts this point of view and propose a general model as an attempt to answer some objections that have been made to this approach, namely (i) the fact that previous published models may need unrealistic assumptions to be applied and (ii) the analytic and algorithmic difficulties in performing the numerical evaluations.

The text is organized as follows. In Section 2 we propose a general approach to built a Markovian graph from the execution structure. The model of the system in operation is described in Section 3. In particular, the failure point process is defined and its analysis is performed in Section 4 where we introduce the analytic and algorithmic tools needed to evaluate it. We obtain the distribution of the number

of failures on a given time interval and an algorithm to compute it. We also study the expectation of this random variable. In Section 5 we discuss on some mathematical properties of the failure process. Some known asymptotic results that can be applied to this context are reviewed, giving some insight on different possible approximations. The last section concludes by some comment.

2 The execution model

The first step in the structural approach to evaluate the dependability of a complex software system is to define an *execution model* taking into account the knowledge of the structure, the potentially available data and the measures that will be calculated. Concerning the first point, the main assumption is that the system can be seen as a set of interacting *components*. This interaction is only made by execution control transfer and, at each instant, control lies in one and only one of the components, which will be called the *active* one. For a discussion on the concept of software component in the context of this paper the reader can see [11]. From this view of the system, a stochastic process representing the active component at each instant is constructed. In order to obtain a tractable mathematical model, the associated stochastic process will be assumed Markov (or eventually semi-Markov.)

Component definition is a user-level task, depending on the system being analyzed, on the possibility of getting the required data, etc. The first idea is to identify the components with the standard software engineering concept of *module*, as it is done for instance in [12],[13],[2],[1],[7] (in other papers, some slightly different choices are done, but the idea is basically the same.) The system structure is then represented by the *call graph* of the set of modules. The main problem with this approach is that the Markovian (or semi-Markovian) assumption may be too much unrealistic. For instance, suppose that module M_i receives the execution control sometimes from module M_j, sometimes from module M_k. In many situations, the conditions under which the control is transferred and the characteristics of the tasks that must be performed by M_i may be completely different in the two cases. If M_i is a state of a Markov process, then the random variable (r.v.) "nth sojourn time in M_i" is independent of the other sojourn times (in M_i and in the other states), does not depend on n and does not depend on the identity of the module from which the control is got. Thus, such a Markovian assumption may be too strong to be acceptable. As another example, consider the case of a module M_i requiring the services of M_j, $M_k, \ldots$. Assume that the execution of M_i begins with some preprocessing code $M_i.1$ followed by calls to some functionalities provided by M_j and M_k, followed by the execution of some ending tasks $M_i.2$ and suppose that the execution times in $M_i.1$ and $M_i.2$ have a very different behavior. If we associate with M_i a state of a Markov process, to estimate the parameter of its exponential sojourn time distribution we will take into account the time of execution of $M_i.1$ and $M_i.2$, which, considered as

r.v., can have very different distributions. Similar considerations can be done on the switching transition probabilities between states, etc.

To avoid (at least, theoretically) the preceding drawbacks, we follow a different approach. Consider a partition of the whole code and call components its elements (the decomposition in modules is then a particular case.) Assume, for instance, that control starts in component C_i. After executing some subset of internal code, the services of component C_j are requested. Once C_j finishes this job, C_i ends in turn and control is transferred to component C_k which, in turn, request immediately the service of C_j. Instead of constructing three states C_i, C_j and C_k, we propose the scheme of Figure 1. The state labeled 'C_i/C_j' represents the execution control in

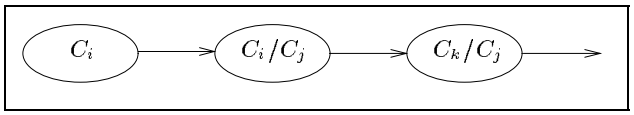

Figure 1: A component requested from two different places

C_j when it has been transferred from C_i and the same for 'C_k/C_j'. Of course, if C_j needs functionalities provided by C_h, we can consider states of the form '$C_i/C_j/C_h$' and '$C_k/C_j/C_h$'.

The main idea is to store in a state information on the *path* (of components) having transferred the control, if it is decided that this is necessary to make the Markovian assumptions realistic. As in previous structural models, when a component C_i transfers the control to either C_j or C_k, we will model the situation by means of probabilistic jumps of the associated stochastic process. Let us point out here that more detailed information on the structure of the system can be used by considering, for instance, particularities of the implementation language, typically to model conditional executions of a set of instructions. For instance, suppose that after executing some code in C_i (denoted by $C_i.1$), control is transferred to C_j; when it is back at component C_i, a `while` loop is (eventually) entered containing a block of code (denoted by $C_i.2$) followed by a second transfer to C_j. This example will allow us to illustrate a supplementary point. The corresponding proposed scheme is shown in Figure 2. As we see, following the same approach, we can differentiate two identical sequences (C_i/C_j) by taking into account where they take place in the text. This can be used if the utilization of the services of C_j is very different when it happens inside or outside the loop. The symbol e denotes the event "the loop is entered" and $\bar{e}$ denotes its complement. Observe that in the constructed stochastic process, not all transitions correspond to a control transfer; for instance, the transition from '$C_i.1$' to '$C_i/C_j(1)$' represents a transfer control but this is not the case for the transition from '$C_i/C_j(2)$' to '$C_i.2$'. We will not discuss on the *flow process modeling* aspect further. The reader can find similar constructions to handle control structures (loops, tests, ...) for instance in [3].

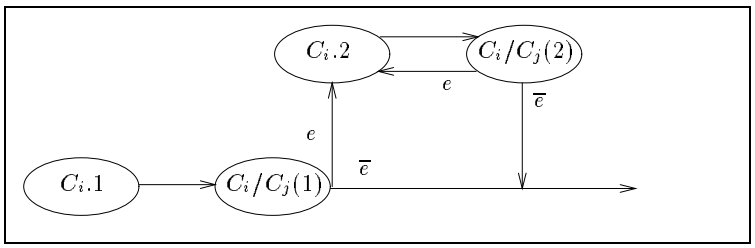

Figure 2: A second illustration of an execution model

Needless to say, it is clear that not all the possible paths must be considered neither all the possible situations illustrated in the examples. Clearly, the opposite must probably be done. We have to identify only a few components leading to a moderate number of states ("labeled paths") because a model is not extremely useful without data and the more complex it is, the more difficult is the measurement task.

To end the section, it remains to underline that the goal here is to give a general description of the approach, not a complete specification of a new formalism. This must depend at least on the type of implementation language(s), the required detail level, the mechanisms that need to be included in the model, etc.

3 The operational model

In this section we develop a model representing the system in operation, which takes into account the eventual failures. Consider the execution process presented in the previous section. It is to be interpreted as the state graph of a homogeneous time-continuous Markovian process X. Its state space is denoted by $\mathcal{M} = \{1, 2, \ldots, M\}$. The process is characterized by its infinitesimal generator, denoted by $Q = (q(i,j))$ $i, j \in \mathcal{M}$, and by its initial distribution $\alpha = (\alpha_1, \ldots, \alpha_M)$. The entries $q(i,j)$ are to be interpreted as the output rate from state i to state j, i.e. the parameter of the exponential distribution of any sojourn time in the state i weighted by the proportion of routing to state j, in absence of any failure phenomenon. The distribution α can be seen as the vector composed by the respective proportion of input nodes selection in the graph by the user, and thus it is called the *user profile*. We will assume X irreducible. This is because either the user does not distinguish any particular run end in its model and considers the continuity of the execution, or because there are states i representing final tasks and, in that case, we add transitions from i to every state j with transition probability α_j.

The second step consists in describing the failure process. Once modeled and added to the execution model, we will obtain the *operational model*. Formally, this consists in constructing a more complex stochastic process starting from X, by considering the fact that failures may happen and their possible effects on the system. We divide the failures in two types. The first one is composed by the *primary failures*

and the occurrence of such a failure is a *primary event*. A primary failure leads to an execution break; the execution is restarted for instance from a checkpoint or perhaps from the beginning. The second class of failures is composed by the *secondary failures* (their occurrences are the *secondary events*) which do not affect the execution process. A secondary failure does not make any change on the execution time in the current state neither on the future of the execution. The main mathematical assumptions are now the following:

- During a sojourn of the execution process in state i, a primary failure occurs with constant rate λ_i. Control is then transferred to some other point, which is modeled by a transition from state i to state j with probability $\alpha(i,j)$. The secondary events (secondary failures) are part of a Poisson process having rate μ_i. Always during a sojourn in state i, the two types of events are independent of each other.

- When control is transferred between two components, failures (primary or secondary) may also happen. In this case they are called *primary interface failures* or *secondary interface failures*. If i and j are two states of the model such that transitions from i to j model control transfers, then a primary (resp. secondary) interface failure occurs with probability $\lambda(i,j)$ (resp. $\mu(i,j)$.) In case of a primary interface failure, the stochastic process jumps to state j, again with probability $\alpha(i,j)$. In order to simplify the technical evaluation of the model, we accept the occurrence simultaneously of the two types of failures (with probability $\lambda(i,j)\mu(i,j)$), with the result that the primary one will be taken into account only. In the model, we set $\lambda(i,j) = \mu(i,j) = 0$ if the transitions from state i to state j do not correspond to control transfers.

- Given a sequence of executed components, the failure processes associated with each state are independent. Also, the interface failure events are independent on each other and on the failures processes occurring during the sojourns.

Observe that we are not interested here in taking into account the delay between two reruns. So, we will neglect the delay between a primary failure and the instant of restart. In a subsequent paper we will discuss these aspects leading, for instance, to availability concerns.

The proposed set of failure processes can be seen as the superposition of the failure processes considered in [12] (or [13]) and [9] (or [2] in a discrete time context). In papers like [12] the considered failures are those called secondary here. We see this as a limitation and our model attempts to avoid it by introducing the primary failures which affect the execution process and by showing how to evaluate the resulting model (see next section.) It must be observed that secondary failures are not exactly a particular case of primary ones with $\alpha(i,i) = 1$. We may want to have in a model both simultaneously, that is, we may want to really differentiate the two

types of failure behavior. Concerning primary failures, observe that the case of a complete reset of the system is handled by setting, for any state i, $\alpha(i,j) = \alpha_j$, that is, by making the jump according to the user profile distribution, independently of the state in which the failure occurs.

4 Model analysis

The main goal of this section is to analyze the distribution of the process (N_t) where N_t is the number of failures (primary or secondary) in the interval $]0,t]$. To do this, let us define the process X_t^* which give the active component in the previous operational model. Following the same approach as in [18] or [14] (where the context is queueing theory), we consider the bi-dimensional time-continuous process (N_t, X_t^*). It follows from the independence between the failure and the execution processes that this is a homogeneous markovian process with state space $\mathbb{N} \times \mathcal{M}$.

Let us denote by $a(i,j)$ (resp. by $d(i,j)$) the transition rate from component i to component j without any failure (resp. with failures.) Formally, we have for all $k \geq 0$:

$$\begin{aligned} a(i,j) &= \lim_{dt \to 0} \frac{1}{dt} \mathbb{P}\{X^*_{t+dt} = j, N_{t+dt} - N_t = 0 | X^*_t = i, N_t = k\} \text{ if } i \neq j, \\ a(i,i) &= 0; \end{aligned}$$

$$d(i,j) = \lim_{dt \to 0} \frac{1}{dt} \mathbb{P}\{X^*_{t+dt} = j, N_{t+dt} - N_t = 1 | X^*_t = i, N_t = k\}.$$

The following expressions for these rates can be derived by listing the different failure events:

$$\begin{aligned} a(i,j) &= q(i,j)(1-\lambda(i,j))(1-\mu(i,j)) \text{ if } i \neq j, \\ a(i,i) &= 0; \end{aligned}$$

$$\begin{aligned} d(i,j) &= [\,\lambda_i + \sum_{k \neq i} q(i,k)\lambda(i,k)\,]\,\alpha(i,j) + q(i,j)[1-\lambda(i,j)]\mu(i,j) \text{ if } i \neq j, \\ d(i,i) &= [\lambda_i + \sum_{k \neq i} q(i,k)\lambda(i,k)]\alpha(i,i) + \mu_i. \end{aligned}$$

Observe that we obtain a Markov process allowing transitions without state changes, that is, loop-transitions. After checking that, for any i, we have

$$\sum_j [a(i,j) + d(i,j)] = \sum_j q(i,j) + \lambda_i + \mu_i$$

and denoting by δ_i this value, let us we define two $M \times M$ matrices D and A where $D = (d(i,j))$ and $A = (a(i,j)) - \mathrm{diag}(\delta_i)$ ($\mathrm{diag}(\delta_i)$ denoting the diagonal matrix

with value δ_i in its (i,i)-entry.) It can be seen that X_t^* is a Markovian homogeneous process defined on $\mathcal{M}$ with initial distribution α and generator $Q^* = A + D$. Since X is irreducible, X^* is irreducible too.

Let us write the Chapman-Kolmogorov (backward) equation for the Markov process (N_t, X_t^*). Denoting the vector $(\mathbb{P}\{N_t = k, X_t^* = j | X_0^* = i, N_0 = 0\})_{j\in\mathcal{M}}$ by $P_i(k,t)$, we have

$$\frac{d}{dt}P_i(k,t) = \sum_{l\in\mathcal{M}} A(i,l)P_l(k,t) + \sum_{l\in\mathcal{M}} D(i,l)P_l(k-1,t). \quad (1)$$

4.1 Particular cases

Before developing the general analysis of the process (N_t), let us recall that a similar model has been used as the arrival process to a queueing system in [14]. Also, when for all $i \in \mathcal{M}$ the probabilities $\alpha(i,j)$ does not depend on i, we have a particular case of the *versatile* point process of Neuts [18],[17].

Let us discuss now some interesting particular cases of the model proposed in this paper.

Example 1. Assume that there is no primary failures, that is, $\lambda_i = 0$ and $\lambda(i,j) = 0$ for all $i,j \in \mathcal{M}$. We obtain the model of Littlewood [12]. Matrices A and D are given by

$$A(i,j) = \begin{cases} q(i,j)(1-\mu(i,j)) & \text{if } i \neq j, \\ -\sum_{j\neq i} q(i,j) - \mu_i & \text{if } i = j, \end{cases}$$

$$D(i,j) = \begin{cases} q(i,j)\mu(i,j) & \text{if } i \neq j, \\ \mu_i & \text{if } i = j. \end{cases}$$

Furthermore, we have $Q^* = Q$ and we retrieve the fact that failures have no influence on the execution process as it was pointed out in Section 3.

Example 2. Another interesting point process is the one obtained by assuming in Littlewood's model that the probability of a secondary failure during a control transfer is 0. In this case, setting $\mu(i,j) = 0$ for all $i,j \in \mathcal{M}$ in the previous expressions, we get $A = Q - \mathrm{diag}(\mu_i)$ and $D = \mathrm{diag}(\mu_i)$. This a Markov Modulated Poisson Process (MMPP), extensively studied for instance in [23].

Example 3. In this case, we consider that there are no secondary failures nor primary failures during a control transfer. Besides, with each primary failure occurrence, we suppose that we have an instantaneous reset of the execution process following the user profile distribution α (that is, the case of $\alpha(i,j) = \alpha_j$.) We obtain a renewal-PH process [16]. Matrix $A = Q - \mathrm{diag}(\lambda_i)$ is the infinitesimal sub-generator

of an absorbing Markov process and $D = -1A^T\alpha$. Then, $Q^* = A - 1A^T\alpha$ is the infinitesimal generator of X^*.

Finally, if we are only concerned in the time to the first failure in the above models, they are mathematically equivalent to the model considered in [9] and this r.v. has a PH distribution [17]. The discrete time counterpart of this case is considered in [2],[21],[22].

4.2 The distribution function of N_t

The distribution function of the number of failures in the time interval $]0,t[$ can be derived from the infinitesimal generator of the bi-dimensional Markov process (N_t, X_t^*). From the Chapman-Kolmogorov equation (1), we deduce the following representation of the (infinite) generator of (N_t, X_t^*) when the states of $\mathbb{N} \times \mathcal{M}$ are ordered as $\{0\} \times \mathcal{M}$, $\{1\} \times \mathcal{M}$, ...

$$\begin{pmatrix} A & D & 0 & \cdots \\ 0 & A & D & \ddots \\ \vdots & \ddots & \ddots & \ddots \end{pmatrix}.$$

So, we can write for all $n \geq 0$,

$$\mathbb{P}\{N_t \geq n\} = 1 - (\alpha, 0, \ldots, 0)e^{Q_n t}1^T, \tag{2}$$

$$\text{where } Q_n = \begin{pmatrix} A & D & 0 & \cdots & & 0 \\ 0 & A & D & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ \vdots & & & \ddots & A & D \\ 0 & \cdots & \cdots & \cdots & 0 & A \end{pmatrix}$$

is a $nM \times nM$ matrix.

To compute the distribution of N_t, we apply the well known uniformization technique [19]. The analysis is similar to the work done in [20] for sojourn times calculations. Let us set

$$P_n \stackrel{\text{def}}{=} I + \frac{1}{\delta}Q_n$$

where δ is the maximal output rate of X^*, that is, $\delta = \max_i \delta_i$. Matrix P_n is a function of matrices $\widehat{A} = I + A/\delta$ and $\widehat{D} = D/\delta$. We can note that $\widehat{A} + \widehat{D} = I + Q^*/\delta$ is the uniformized matrix of the infinitesimal generator Q^* of X^*. Noting that for $h \leq n-1$, $(\alpha, 0, \cdots, 0)Q_n^h 1^T = 0$, we derive from (2) and for $n \geq 1$,

$$\mathbb{P}\{N_t \geq n\} = \sum_{h=n}^{+\infty} e^{-\delta t}\frac{(\delta t)^h}{h!}\left(1 - \alpha x^T(n,h)\right). \tag{3}$$

where $x^T(n,h)$ denote the vector composed by the first n entries of $P_n^h\, 1^T$.

The upper triangular pattern of matrix P_n provide us the following recursive scheme for computing $x^T(n,h)$:

$$\begin{aligned} x^T(n,h) &= \widehat{A}x^T(n,h-1) + \widehat{D}x^T(n-1,h-1), \\ x^T(0,h) &= 0 \quad \forall h \geq 0, \\ x^T(n,0) &= 1 \quad \forall n \geq 1. \end{aligned} \tag{4}$$

The computation of (3) reduces to the following steps. For fixed n,

(a) choose the tolerance error ε;

(b) compute H such that $\sum_{h=H+1}^{+\infty} e^{-\delta t}\frac{(\delta t)^h}{h!} < \varepsilon$;

(c) compute the entries $x^T(k,h)$ for $h = k, ..., k+H-n$ with (4). This is illustrated in Figure 3: the cell (k,h) is deduced from the cells $(k,h-1)$ and $(k-1,h-1)$ using the first relation of (4). In other words, the computations are made in a column by column manner and only concern the $(nH - n^2)$ bold cells in Figure 3.

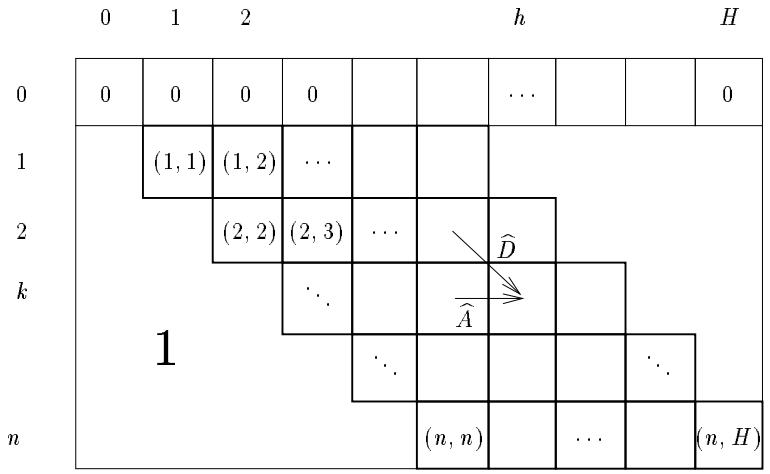

Figure 3: The contents of cell (k,h) is the vector $x^T(k,h)$

So, we need to store at most n M-dimensional vectors. Note that we can get, with this algorithm, the probabilities $\mathbb{P}\{N_t \geq 1\}, \ldots, \mathbb{P}\{N_t \geq n\}$ by computing all the rows of the table in Figure 3.

Example 3 (continued.) In the interesting particular case of a renewal-PH process, it can be better to perform the computation of the cells in Figure 3 in a row by row manner. Indeed, we have $D = -1A^T\alpha$ and

$$\widehat{D}x^T(k-1,h-1) = -\frac{1}{\delta}1A^T\alpha x^T(k-1,h-1).$$

Therefore, at step k, we need only to store the $(H-n+1)$ **scalars** $\alpha x^T(k-1,h)$, $h = k-1,k,\ldots,k-1+H-n$. Once cell $x^T(k,h)$ is computed, the scalar $\alpha x^T(k,h-1)$ is evaluated and stored in place of the scalar $\alpha x^T(k-1,h-1)$.

4.3 Computing the expectation of N_t

From (3) we get

$$\begin{aligned} E[N_t] &= \sum_{n=1}^{+\infty} \mathbb{P}\{N_t \geq n\} \\ &= \delta t - \sum_{h=1}^{+\infty} e^{-\delta t}\frac{(\delta t)^h}{h!}\left[\sum_{n=1}^{h}\alpha x^T(n,h)\right] \end{aligned} \tag{5}$$

where $\sum_{n=1}^{h}\alpha x^T(n,h)$ is the cumulative sum of the upper cells of the hth column in Figure 3. Therefore, an initial estimation of the expectation can be deduced from the computation of the distribution function. However, its quality depends basically on the value of δt. In general, the same tolerance error ε for the expectation requires to consider higher values of H than for the distribution function. The algorithm is just the computation of the sum in the right hand side of (5) and can be resumed as

(a) choose the tolerance error ε;

(b) compute H such that $\delta t\sum_{h=H}^{+\infty} e^{-\delta t}\frac{(\delta t)^h}{h!} < \varepsilon$;

(c) for each $h = 1,\ldots,H$, compute the column of scalars $\alpha x^T(n,h)$ with (4) for $n = 1,\ldots,h$.

5 Some properties of the failure point process

The following formulae can be derived from the ones given in [17], using the simpler formalism of matrices A and D. They are useful in pointing out the asymptotic behavior of the point process N_t. In particular, they give some insight into the Poissonnian approximation given, for instance, in [12]. The first relation concerns the expectation of N_t. If we denote by $(N_i(k,t))$ the column vector $(\mathbb{P}\{N_t = k | N_0 = 0, X_0^* = i\})_{i\in\mathcal{M}}$,

we can verify, from the Chapman-Kolmogorov relations (1), that these quantities satisfy the following ordinary differential equation:

$$\frac{d}{dt}\left(N_i(k,t)\right) = A\left(N_i(k,t)\right) + D\left(N_i(k-1,t)\right). \tag{6}$$

Multiplying each equation by k and summing on k, we obtain another ordinary differential equation for the vector of expectations $(E_i[N_t]) = \sum_{k\geq 1} k\left(N_i(k,t)\right)$:

$$\frac{d}{dt}\left(E_i[N_t]\right) = Q^*\left(E_i[N_t]\right) + D1^T.$$

The solution to this last equation is given by

$$\left(E_i[N_t]\right) = \int_0^t e^{Q^* u} du \;\; D1^T. \tag{7}$$

Reporting in (7) the well known expression of the fundamental matrix associated with a continuous time Markov process with generator Q^*

$$\int_0^t e^{Q^* u} du \;-\; (1^T\pi^*)t = (I - e^{Q^* t})(1^T\pi^* - Q^*)^{-1} \tag{8}$$

we obtain

$$\left(E_i[N_t]\right) = \{(1^T\pi^*)t + (I - e^{Q^* t})(1^T\pi^* - Q^*)^{-1}\}D1^T.$$

Introducing the initial distribution and deconditioning, we get

$$\begin{aligned} E[N_t] \;\; = \;\; & \lambda^* t + (\alpha - \pi^*)(1^T\pi^* - Q^*)^{-1} D\, 1^T \\ & + (\pi^* - \alpha e^{Q^* t})(1^T\pi^* - Q^*)^{-1} D\, 1^T \end{aligned} \tag{9}$$

where $\lambda^* = \pi^* D\, 1^T = \sum_i \pi_i^* (\sum_j D_{ij})$ is called the *fundamental rate* of the failure process and π^* is the stationary distribution associated with the irreducible generator Q^*.

In the particular case of the process considered in Example 1, we have

$$\lambda^* = \sum_i \pi_i \left[\sum_j q(i,j)\mu(i,j) + \mu_i \right],$$

where π is the stationary distribution associated with the irreducible execution process X.

Relation (9) shows that $E[N_t]$ is the sum of a linear term in t and of a "perturbation" term depending on the transient behavior (state probabilities) of X_t^*. In particular, asymptotically as t tends to infinity, we have

$$E[N_t] = \lambda^* t + (\alpha - \pi^*)(1^T\pi^* - Q^*)^{-1} D\, 1^T + o(1). \tag{10}$$

The convergence rate of the state probabilities of X_t^* to the distribution π^* determines the quality of the linear approximation in (10).

Remark. Relation (9) can be exploited for the computation of $E[N_t]$ knowing the vectors π^* and $(1^T\pi^* - Q^*)^{-1} D\, 1^T$. This evaluation only involves the state probabilities $\alpha e^{Q^* t}$ and it can be done for instance by the uniformization technique [6]. Another way [17] is to note that the vector $v(t) = (1^T\pi^* - e^{Q^* t})(1^T\pi^* - Q^*)^{-1} D\, 1^T$ is a solution to the ordinary differential system

$$\begin{array}{rcl} v(t) & = & Q^* v(t) \\ v(0) & = & (I - 1^T\pi^*)(1^T\pi^* - Q^*)^{-1} D\, 1^T \end{array}$$

which can be numerically integrated.

From relations (9) and (10), the condition $\alpha = \pi^*$ is necessary for a Poissonnian behavior (with parameter λ^*) of our point process. This can be interpreted as the observation of the stationary version of the point process. Then, we have $E[N_t] = \lambda^* t$, for all $t \geq 0$, and λ^* represent the expected number of failures per time unit.

If we investigate the variance of N_t in the stationary version of our point process, the first step consists in computing the second factorial moment $E_{\pi^*}[N_t(N_t - 1)] = \sum_{k \geq 2} k(k-1)\ \pi^*(N_i(k,t))$. The following ordinary differential equation can be derived by multiplying (6) by $k(k-1)$ and summing:

$$\frac{d}{dt} E_{\pi^*}[N_t(N_t - 1)] = 2\pi^* D\,(E_i[N_t])\,.$$

Solving it allows us to obtain the expression

$$E_{\pi^*}[N_t(N_t - 1)] = 2\pi^* D \int_0^t \int_0^s e^{Q^* u} du\, ds D 1^T.$$

The second factorial moment follows using (8) in the previous relation

$$\begin{array}{rcl} E_{\pi^*}[N_t(N_t - 1)] & = & (\lambda^*)^2 t^2 + \{2\pi^* D(1^T\pi^* - Q^*)^{-1} D\, 1^T - 2(\lambda^*)^2\} t \\ & & -2\pi^* D(I - e^{Q^* t})(1^T\pi^* - Q^*)^{-2} D\, 1^T. \end{array}$$

Finally, using the relation $\sigma^2(t) = E_{\pi^*}[N_t(N_t - 1)] + \lambda^* t - (\lambda^*)^2 t^2$ we have

$$\begin{array}{rcl} \sigma^2(t) & = & \left\{\lambda^* + 2\pi^* D(1^T\pi^* - Q^*)^{-1} D\, 1^T - 2(\lambda^*)^2\right\} t \\ & & -2\pi^* D(I - 1^T\pi^*)(1^T\pi^* - Q^*)^{-2} D\, 1^T \\ & & -2\pi^* D(1^T\pi^* - e^{Q^* t})(1^T\pi^* - Q^*)^{-2} D\, 1^T. \end{array}$$

As for the expectation, a linear asymptote appears when t goes to infinity, plus a "perturbation" proportional to $(1^T\pi^* - e^{Q^* t})$, so decreasing to 0. However, even with

the stationary version of process X_t^*, a priori the variance is far from the Poisson process one.

The last formula concerns the structure of the covariance C_{π^*} of the stationary version of N_t : if $C_{\pi^*}(t,t';t_0) \stackrel{\text{def}}{=} E[N_t(N_{t'+t+t_0} - N_{t+t_0})] - \lambda^{*2}tt'$ then

$$C_{\pi^*}(t,t';t_0) = \pi^* D(I - e^{Q^*t})e^{Q^*t_0}(I - e^{Q^*t'})(1^T\pi^* - Q^*)^{-2}D\,1^T.$$

As t_0 tends to infinity then $C_{\pi^*}(t,t';t_0) = o(1)$ which implies that the number of failures in two disjoint time intervals are asymptotically non correlated.

Example 2 (continued.) Finally, let us discuss the asymptotic approximation of N_t by a Poisson process when the matrix D associated with Littlewood's model (Example 1) tends to 0. A necessary (but not sufficient) condition is to assume X^* (equal to the execution process in this case) in its stationary state. For the simpler MMPP model, we have $A = Q - \text{diag}(\mu_i)$ and if for all i we have $\mu_i \to 0$, then

$$\begin{aligned}
\mathbb{P}\{N_t = 0\} &= \alpha e^{At}\,1^T, \\
&\approx 1 - \alpha \int_0^t e^{Qs}ds\,\text{diag}(\mu_i)\,1^T, \\
&\approx 1 - (\sum_i \pi_i\mu_i)t + \left\{(\alpha - \alpha e^{Qt})(1^T\pi - Q)^{-1}\right\}\,\text{diag}(\mu_i)\,1^T.
\end{aligned}$$

In this case we have the first order development of the exponential with parameter $\lambda^* = \sum_i \pi_i\mu_i$ only if $\alpha = \pi$.

It is intuitively clear that if for all state i the failure rate μ_i tends to 0, then the time to the next event will be stochastically increasing and the execution process will be closer of its stationary state at the time of a failure occurrence. This stochastical growth can be formally checked in the above expression concerning the MMPP model. However, the only condition that $\mu_i \to 0$ for all i has no effect on the execution process because the (secondary) failure process has no influence on the execution one.

6 Conclusion

We have discussed in this paper the counting process of a structural markovian software reliability model. Our approach makes the Markovian assumptions better adapted to the context of software reliability evaluated by means of structural models and our model is shown to be richer than previous published ones. In particular, we derive the distribution function and the expectation of the number of failures on an interval. Note that the stochastic phenomenon of reliability growth of some component can be added to the proposed model using the transformation approach as developed in [10],[5]. (This method is not discussed here.) It can be also argued that

the primary failure rate must be decreasing during a sojourn time in a component. The feeling is that the longer the sojourn time, the less likely the failure occurrence. In this case, the classical "method of stages", widely used for instance in queuing theory, can simulate this feature.

References

[1] N. Ashrafi and F. Zahedi. Software reliability allocation based on structure, utility, price, and cost. *IEEE Trans. Software Eng.*, 17(4):345–356, 1991.

[2] R.C. Cheung. A user-oriented software reliability model. *IEEE Trans. Software Eng.*, 6(2):118–125, 1980.

[3] N.E. Fenton. *Software metrics : a rigorous approach.* Chapman and Hall, 1991.

[4] A. Iannino and al. *Software Reliability: Measurement, Prediction, Application.* McGraw-Hill International Editions, Computer Science Series, 1987.

[5] M. Kaâniche. *Modèle Hyperexponentiel en Temps Continu et en Temps Discret pour l'Evaluation de la Croissance de la Sûreté de Fonctionnement.* PhD thesis, LAAS, 7 Avenue du Colonel Roche, 31077 TOULOUSE Cedex (France), 1992.

[6] E.P.C. Kao. Computing the phase-type renewal and related functions. *Technometrics*, 30(1):87–93, 1988.

[7] P. Kubat. Assessing reliability of modular software. *Opns Res. Letters*, 8:35–41, 1989.

[8] J.C. Laprie. *Dependability: Basic Concepts and Terminology.* Springer-Verlag, Vienna, 1992.

[9] J.C. Laprie. Dependability evaluation of software systems in operation. *IEEE Trans. Software Eng.*, SE-16(6):701–714, 1984.

[10] J.C. Laprie and al. The KAT (knowledge-action-transformation) approach to the modeling and evaluation of reliability and availability growth. *IEEE Trans. Software Eng.*, 17(4):370–382, 1991.

[11] J.C. Laprie and K. Kanoun. X-ware reliability and availability modeling. *IEEE Trans. Software Eng.*, 18(2):130–147, 1992.

[12] B. Littlewood. A reliability model for systems with markov struture. *Appl. Statist.*, 24(2):172–177, 1975.

[13] B. Littlewood. Software reliability model for modular program structure. *IEEE Trans. Reliability*, R-28(3):241–246, 1979.

[14] D.M. Lucantoni and al. A single-server queue with server vacations and a class of non-renewal arrival processes. *Adv. Appl. Prob.*, 22:676–705, 1990.

[15] Y. Masuda and al. A statistical approach for determining release time of software system with modular structure. *IEEE Trans. Reliability*, 38(3):365–372, 1988.

[16] M.F. Neuts. *Matrix-Geometric Solutions in Stochastic Models : An Algorithmic Approach.* The John Hopkins University Press., 1981.

[17] M.F. Neuts. *Structured Stochastic Matrices of M/G/1 Type and Their Applications.* Marcel Dekker Inc., New-York and Basel, 1989.

[18] M.F. Neuts. A versatile markovian point process. *J. Appl. Prob.*, 16:764–779, 1979.

[19] S.M. Ross. *Stochastic Processes.* John Wiley and Sons, 1983.

[20] G. Rubino and B. Sericola. Interval availability analysis using operational periods. *Performance Evaluation*, 14:257–272, 1992.

[21] K. Siegrist. Reliability of systems with markov transfer of control. *IEEE Trans. Software Eng.*, 14(7):1049–1053, 1988.

[22] K. Siegrist. Reliability of systems with markov transfer of control, II. *IEEE Trans. Software Eng.*, 14(10):1478–1480, 1988.

[23] D.L. Snyder. *Random Point Processes.* Wiley-Interscience Publication, 1975.

[24] M. Xie. *Software Reliability Modelling.* World Scientific Publishing, UK office: 73 Lynton Mead, Totteridge, London N20 8DH, 1991.

Unité de recherche INRIA Lorraine, Technôpole de Nancy-Brabois, Campus scientifique,
615 rue de Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 46 avenue Félix Viallet, 38031 GRENOBLE Cedex 1
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
ISSN 0249-6399